PDO trong PHP - Khái niệm và những thao tác cơ bản

Nguồn bài viết : Viblo

Nếu bạn là một PHP Developer, chắc hẳn bạn đã rất quen thuộc với việc truy xuất Database (Cơ sở dữ liệu) bằng các extensions MySQL và MySQLi. Từ PHP 5.1 ta có một cách thức tối ưu hơn đó là sử dụng PHP Data Objects. PDO cung cấp các cơ chế Prepared Statements, Stored Procedures và giúp bạn thao tác với database thông qua các Object (đối tượng) làm cho công việc trở nên hiệu quả, dễ dàng hơn.

So sánh PDO và MySQLi

	PDO	MySQLi
Database hỗ trợ	Hơn 12 loại	Chỉ hỗ trợ MySQL
API	Hướng đối tượng (OOP)	Hướng đối tượng (OOP) - Hướng thủ tục (Procedural)
Kết nối Database	Dễ dàng	Dễ dàng
Đặt tên tham số	Có	Không
Object Mapping	Có	Có
Prepared Statements	Có	Không
Hiệu năng	Cao	Cao
Stored Procedures	Có	Có

1. Giới thiệu PDO - PHP Data Objects

PHP Data Objects (PDO) là một lớp truy xuất cơ sở dữ liệu cung cấp một phương pháp thống nhất để làm việc với nhiều loại cơ sở dữ liệu khác nhau. Khi làm việc với PDO bạn sẽ không cần phải viết các câu lệnh SQL cụ thể mà chỉ sử dụng các phương thức mà PDO cung cấp, giúp tiết kiệm thời gian và làm cho việc chuyển đổi Hệ quản trị cơ sở dữ liệu trở nên dễ dàng hơn, chỉ đơn giản là thay đổi Connection String (chuỗi kết nối CSDL).

Bạn chỉ cần nắm rõ API mà PDO cung cấp là có thể làm việc được với nhiều Hệ quản trị cơ sở dữ liệu khác nhau như MySQL, SQLite, PostgreSQL, Microsoft SQL Server,... và có thể dễ dàng chuyển đổi chúng.

Các Hệ quản trị cơ sở dữ liệu (Database Management System) mà PDO hỗ trợ gồm có:

Tên driver	DBMS
PDO_CUBRID	Cubrid
PDO_DBLIB FreeTDS	Microsoft SQL Server / Sybase
PDO_FIREBIRD	Firebird
PDO_IBM	IBM DB2
PDO_INFORMIX	IBM Informix Dynamic Server
PDO_MYSQL	MySQL 3.x/4.x/5.x
PDO_OCI	Oracle Call Interface
PDO_ODBC	ODBC v3 (IBM DB2, unixODBC and win32 ODBC)
PDO_PGSQL	PostgreSQL
PDO_SQLITE	SQLite 3 and SQLite 2

Tên driver	DBMS
PDO_SQLSRV	Microsoft SQL Server / SQL Azure
PDO_4D	4D

2. Kết nối cơ sở dữ liệu

Mỗi DBMS sẽ có các phương thức kết nối khác nhau (có loại cần Username, Password, đường dẫn đới Database, Port, có loại không). Connection String của các DBMS phổ biến hầu hết đều có dạng như sau:

```
$conn = new PDO('mysql:host=localhost;dbname=izlearn', $username,
$password);
```

Với mysql là tên của DBMS, localhost có ý nghĩa database được đặt trên cùng server, izlearn là tên của database. $username và $password là 2 biến chứa thông tin xác thực.

Đối với SQLite, DBMS này không có cơ chế xác thực bằng Username và Password mà chỉ đơn giản là đường dẫn tới file dữ liệu:

```
$conn = new PDO("sqlite:your/database/path/izlearn.db");
```

Đây là lúc để bạn quên đi Connection String lỗi thời mysql_connect('localhost', 'username', 'password') or die('Could not connect: ' . mysql_error()); Hiện vẫn còn rất nhiều bài viết ở Việt Nam hướng dẫn người mới sử dụng cách kết nối CSDL dạng này vì họ cho rằng nó đơn giản. Thực ra họ chỉ đang dẫn bạn đi về quá khứ mà thôi. Để ngắt kết nối khi không cần thao tác với database nữa, các bạn chỉ cần sét biến $conn về null;

```
$conn = null;
```

3. Insert và Update

Thêm mới (insert) và cập nhật (update) dữ liệu là những hoạt động cơ bản khi thao tác với database. Với PDO, mỗi hoạt động insert hay update được thực hiện qua 3 quá trình sử dụng cơ chế Prepared Statement

- Prepare statement: Chuẩn bị một câu lệnh SQL làm khung/mẫu được gọi là Prepared Statement với các Placeholder (có thể hiểu placeholder đóng vai trò như tham số của các phương thức khi bạn khai báo hàm)
- Bind params: Gắn giá trị thực vào các placeholder (tương tự như khi bạn truyền giá trị vào các tham số của phương thức)
- Execute: Thực thi câu lệnh.

Prepared Statement

Có 2 loại Placeholder trong Prepared Statement là Placeholder không định danh (Unnamed Placeholder) và Placeholder định danh (Named Placeholder) như ví dụ sau:

```
$stmt = $conn->prepare('INSERT INTO users (name, email, age) values (?, ?, ?)');

$stmt = $conn->prepare('INSERT INTO users (name, email, age) values (:name, :mail, :age)');
```

Dòng lệnh thứ nhất sử dụng Placeholder không định danh là các dấu hỏi - ?. Dòng lệnh thứ 2 sử dụng Placeholder định danh: :name, :mail, :age (lưu ý dấu hai chấm và placeholder không nhất thiết phải giống tên column). Sau đây là toàn bộ quá trình Insert và Update sử dụng 2 loại Placeholder nêu trên.

Unnamed Placeholder

```
//Khởi tạo Prepared Statement từ biến $conn ở phần trước

$stmt = $conn->prepare('INSERT INTO users (name, email, age) values (?, ?, ?)');


//Gán các biến (lúc này chưa mang giá trị) vào các placeholder theo thứ tự tương ứng

$stmt->bindParam(1, $name);

$stmt->bindParam(2, $mail);

$stmt->bindParam(3, $age);


//Gán giá trị và thực thi

$name = "Vu Hoang Lam"<

$mail = "lamvh@live.com";

$age = 22;

$stmt->execute();


//Gán những giá trị khác và tiếp tục thực thi

$name = "Nguyen Van A";

$mail = "nva@live.com";

$age = 23;

$stmt->execute();
```

Như các bạn thấy ta chỉ cần khởi tạo Prepared Statement một lần và có thể sử dụng lại nhiều lần. Với mỗi column - placeholder ta phải thực hiện gán tham số một lần, điều này sẽ không sao với những table có ít column như ví dụ trên, nhưng sẽ rất bất tiện nếu bảng có nhiều table, rất may mắn ta

có cách khác để làm việc này, đó là lưu toàn bộ giá trị vào trong một mảng và truyền mảng này vào phương thức execute(), cụ thể như sau:

```
$stmt = $conn->prepare('INSERT INTO users (name, email, age) values (?, ?, ?)');

$data = array('Vu Hoang Lam', 'lamvh@live.com', 22);


//Phương thức execute() dưới đây sẽ gán lần lượt giá trị trong mảng vào các Placeholder theo thứ tự

$stmt->execute($data);
```

Named Placeholder

Đối với Named Placeholder, cách thực hiện cũng khá tương đồng với Unnamed Placeholder, chỉ khác là ta không dùng thứ tự placeholder để gán giá trị (bind) mà dùng chính tên của placeholder:

```
//Khởi tạo Prepared Statement từ biến $conn ở phần trước

$stmt = $conn->prepare('INSERT INTO users (name, email, age) values (:name, :mail, :age)');


//Gán các biến (lúc này chưa mang giá trị) vào các placeholder theo tên của chúng

$stmt->bindParam(':name', $name);

$stmt->bindParam(':mail', $mail);

$stmt->bindParam(':age', $age);


//Gán giá trị và thực thi

$name = "Vu Hoang Lam";

$mail = "lamvh@live.com";
```

```
$age = 22;

$stmt->execute();
```

Các bạn cũng có thể sử dụng mảng để rút gọn:

```
//Lưu ý: Không cần thiết phải sử dụng dấu hai chấm cho các key

$data = array('name' => 'Vu Hoang Lam', 'mail' => 'lamvh@live.com', 'age'
=> 22);

Một mẹo hữu ích khác khi sử dụng Named Placeholder đó là insert Object


class $user

{

    public $name;

    public $mail;

    public $age;

}


$person = new $user();

$person->name = 'Vu Hoang Lam';

$person->mail = 'lamvh@live.com';

$person->age = 22;


$stmt = $conn->prepare('INSERT INTO users (name, email, age) values
(:name, :mail, :age)');


$stmt->execute((array)$person);
```

Ở dòng cuối cùng, tôi đã thực hiện "ép kiểu" (cast) Object $person thành array để truyền vào phương thức execute();

Việc sử dụng Prepared Statement sẽ giúp bạn tránh được SQL Injection, tôi sẽ đi sâu giải thích vấn đề này trong một bài viết khác.

4. Select Data - "Đọc" dữ liệu từ database

Khi đọc dữ liệu từ database, PDO sẽ trả về dữ liệu theo cấu trúc mảng (array) hoặc đối tượng (object) thông qua phương thức fetch(). Bạn nên thiết lập trước cấu trúc dữ liệu trước khi gọi phương thức này, PDO hỗ trợ các tuỳ chọn sau:

- PDO::FETCH_ASSOC: Trả về dữ liệu dạng mảng với key là tên của column (column của các table trong database)
- PDO::FETCH_BOTH (default): Trả về dữ liệu dạng mảng với key là tên của column và cả số thứ tự của column
- PDO::FETCH_BOUND: Gán giá trị của từng column cho từng biến đã khởi tạo trước đó qua phương thức bindColumn()
- PDO::FETCH_CLASS: Gán giá trị của từng column cho từng thuộc tính (property/attribute) của một lớp Class theo tên column và tên thuộc tính.
- PDO::FETCH_INTO: Gán giá trị của từng column cho từng thuộc tính của một Class Instance (thể hiện của một lớp)
- PDO::FETCH_LAZY: Gộp chung PDO::FETCH_BOTH/PDO::FETCH_OBJ
- PDO::FETCH_NUM: Trả về dữ liệu dạng mảng với key là số thứ tự của column
- PDO::FETCH_OBJ: Trả về một Object của stdClass (link is external) với tên thuộc tính của Object là tên của column.

Trong thực tế, chúng ta chỉ thường dùng 3 kiểu fetch đó là: FETCH_ASSOC, FETCH_CLASS và FETCH_OBJ. Để thiết lập cấu trúc dữ liệu (Fetch Style hay Fetch Mode) trước khi fetch ta dùng câu lệnh sau:

```
$stmt->setFetchMode(PDO::FETCH_ASSOC);
```

Hoặc nếu muốn bạn cũng có thể thiết lập kiểu fetch khi gọi hàm fetch():

```
$stmt->fetch(PDO::FETCH_ASSOC);
```

FETCH_ASSOC

Kiểu fetch này sẽ tạo ra một mảng kết hợp lập chỉ mục theo tên column (nghĩa là các key của mảng chính là tên của column), tương tự như khi ta dùng MySQL/MySQLi Extension.

```
//Tạo Prepared Statement
$stmt = $conn->prepare('SELECT email, age from users WHERE name = :name');


//Thiết lập kiểu dữ liệu trả về
$stmt->setFetchMode(PDO::FETCH_ASSOC);


//Gán giá trị và thực thi
$stmt->execute(array('name' => 'a'));


//Hiển thị kết quả, vòng lặp sau đây sẽ dừng lại khi đã duyệt qua toàn bộ kết quả
while($row = $stmt->fetch()) {
    echo $row['name'] , '\n';
    echo $row['email'] , '\n';
    echo $row['age'] , '\n';
```

```
}
```

FETCH_OBJ

Kiểu fetch này trả về một Object của stdClass cho mỗi row của kết quả.

```
//Tạo Prepared Statement
$stmt = $conn->prepare('SELECT email, age from users WHERE name = :name');


//Thiết lập kiểu dữ liệu trả về
$stmt->setFetchMode(PDO::FETCH_OBJ);


//Gán giá trị và thực thi
$stmt->execute(array('name' => 'a'));


//Hiển thị kết quả, vòng lặp sau đây sẽ dừng lại khi đã duyệt qua toàn bộ kết quả trả về
while($row = $stmt->fetch()) {
    echo $row->name , '\n';
    echo $row->email , '\n';
    echo $row->age , '\n';
}
```

FETCH_CLASS

Kiểu fetch này cho phép bạn đưa kết quả vào Object của một Class mà bạn chỉ định. Khi sử dụng FETCH_CLASS, thuộc tính của class sẽ được gán giá trị trước khi constructor của class đó được gọi (phải chú ý vì điều này rất quan

trọng). Nếu không có thuộc tính khớp với tên của một column bất kỳ thì thuộc tính đó sẽ được tự động tạo ra (public).

Giả sử table users có một ta đã có Class User được định nghĩa như sau:

```
class User {

    public $name;

    public $email;

    public $isAdmin = 'No';


    function __construct() {

        if ($this->name == 'Vu Hoang Lam')

            $this->isAdmin = 'Yes';

    }

}
```

Khi query data từ database sử dụng đoạn code sau:

```
//Tạo Prepared Statement

$stmt = $conn->prepare('SELECT email, age from users WHERE name = :name');


//Thiết lập kiểu dữ liệu trả về, chỉ định dữ liệu được đưa vào object của
class User

$stmt->setFetchMode(PDO::FETCH_CLASS, 'User');


//Gán giá trị và thực thi

$stmt->execute(array('name' => 'a'));
```

```
//Hiển thị kết quả, vòng lặp sau đây sẽ dừng lại khi đã duyệt qua toàn bộ
kết quả trả về

while($obj = $stmt->fetch()) {

    echo $obj->email;

    echo $obj->isAdmin;

}
```

Vì constructor được gọi sau khi thuộc tính $name được gán bằng Vu Hoang Lam nên isAdmin sẽ mang giá trị Yes. Nếu muốn constructor của class được gọi trước khi các thuộc tính được gán giá trị, bạn phải sử dụng thêm PDO::FETCH_PROPS_LATE. Cách sử dụng như sau:

```
//Hãy thử sử dụng kiểu fetch trên và so sánh kết quả hiển thị

$stmt->setFetchMode(PDO::FETCH_CLASS | PDO::FETCH_PROPS_LATE, 'User');
```

Nếu cần truyền các tham số cho constructor của class thông qua phương thức fetch(), các bạn có thể đặt chúng trong một array theo thứ tự tương ứng cụ thể như sau:

```
$stmt->setFetchMode(PDO::FETCH_CLASS, 'User', array('param1', 'param2',
'param3'));
```

Exceptions - Xử lý ngoại lệ

PDO dùng các Exceptions để xử lý các lỗi phát sinh khi làm việc với database, vì thế tất cả những gì bạn làm với PDO nên được đặt trong một try/catch block. PDO cung cấp 3 chế độ xử lý lỗi (Error Mode) được thiết lập thông qua phương thức setAttribute():

```
$conn->setAttribute( PDO::ATTR_ERRMODE, PDO::ERRMODE_SILENT );

$conn->setAttribute( PDO::ATTR_ERRMODE, PDO::ERRMODE_WARNING );
```

```
$conn->setAttribute( PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION );
```

PDO::ERRMODE_SILENT

Đây là chế độ xử lý lỗi mặc định của PDO, khi gặp mốt lỗi bất kỳ, PDO sẽ im lặng (silent) và chương trình vẫn tiếp tục chạy. Bạn có thể lấy mã lỗi và thông tin về các lỗi đã xảy ra qua PDO::errorCode() và PDO::errorInfo()

PDO::ERRMODE_WARNING

Ở chế độ này khi gặp phải lỗi PDO sẽ ném ra một PHP Warning, chương trình sẽ tiếp tục chạy.

PDO::ERRMODE_EXCEPTION

Đây là mode mà bạn nên sử dụng nhất, khi đặt trong một try/catch block sẽ giúp bạn kiểm soát các lỗi phát sinh một cách uyển chuyển và giấu các thông báo lỗi có thể khiến Attacker khai thác hệ thống của bạn.

```
try {
    $stmt = new PDO('mysql:host=localhost;dbname=izlearn', 'lamvh', 'talapassday');
    $stmt->setAttribute( PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION );

    //Sai cú pháp, FORM thay vì FROM
    $stmt->prepare('SELECT name FORM people');
}
catch(PDOException $e) {
    echo "ERROR! Co loi xay ra voi PDO";
    file_put_contents('PDOErrors.txt', $e->getMessage(), FILE_APPEND);
```

```
}
```

Đoạn code trên sẽ ghi thông báo lỗi vào một file text với tên PDOErrors.txt

Một số phương thức hữu ích khác

```
$conn->lastInsertId();
```

Phương thức trên trả về Auto Incremented ID của rows được thêm gần nhất.

```
$conn->exec('DELETE FROM users WHERE uid = 1');
```

Đối với các lệnh SQL không có dữ liệu trả về, và không cần thiết phải truyền tham số thì có thể sử dụng phương thức exec(). Phương thức này sẽ trả về số lượng row bị tác động sau khi thực hiện câu lệnh. Như ví dụ trên sẽ trả về số lượng row bị xoá.

```
$conn = $DBH->quote($foo);
```

Phương thức quote() sẽ giúp bạn thêm dấu nháy cho một string để string đó an toàn khi sử dụng để truy vấn, nếu bạn không muốn sử dụng Prepared Statement.

```
$stmt->rowCount();
```

Phương thức rowCount() trả về số lượng row bị tác động sau khi thực hiện các thao tác DELETE, INSERT và UPDATE. Dùng rowCount() cho thao tác SELECT có thể sẽ trả về kết quả không đúng với một số loại database.

Tham khảo: http://www.izlearn.com/